AMENDMENTS TO THE SPECIFICATION

Please amend paragraph [0020] of the specification at page 4 as follows:

[0020] FIGS. 15A ~~and 15B~~ 15C are block diagrams, illustrating computation of motion candidates, in accordance with one embodiment of the invention.

Please amend paragraph [0036] of the specification at page 8 as follows:

[0036] In one embodiment, network processor 400 is, for example, implemented within processor 100 of FIG. 1, such that processor 100 of FIG. 1 implements a multi-threaded micro-architecture. In one embodiment, a general framework of automatic thread partitioning for traditional network packet processing applications may be performed on any computer architecture, including the following features: (1) ~~provides~~ a multi-threaded architecture; (2) ~~provides~~ thread local storage access by each thread exclusively, as well as global storage access by all threads; and (3) ~~provides~~ inter-thread communication and synchronization mechanism signals. Procedural methods for implementing embodiments of the invention are now described.

Please amend paragraph [0048] of the specification at page 12 as follows:

[0048] At process block 550, it is determined whether hoist instructions are no longer detected from the identified motion candidate instructions. Until such is the case, hoist instructions detected from the identified motion candidate instructions are hoisted within the basic blocks of the CFG loop at process block 534. At process block 552, ~~process block 552~~ motion candidate instructions are ~~hoisted~~ reordered in a source basic block of the CFG loop according to a dependence graph of the sequential application program using intra-block hoisting of identified motion candidate instructions as shown in FIG. 15A-15C. As ~~descried~~ described herein, a dependence graph is constructed to illustrate data dependence of a PPS loop body to provide information about data dependence. Hence, hoisting or sinking any motion candidate instructions cannot violate data dependence ~~on~~ of the original program. As ~~descried~~ described herein, a dependence graph illustrates data dependence between nodes and control dependence between nodes.

Please amend paragraph [0059] of the specification at page 14 as follows:

[0059] Referring again to FIG. 12, once motion candidates are identified, at process block 558, a sink queue is initialized with basic blocks of the CFG loop. In one embodiment, the basis blocks are ordered within the sink queue based on a topological order in the CFG loop. At process block 560, motion candidate instructions are sunk among the ~~basing~~ basic blocks until sinking instructions are no longer detected from the motion candidate instructions at process block 574. At process block 576, motion candidate instructions are ~~sunk~~ reordered within basic blocks that contain ADVANCE operations according to the dependence graph of the sequential application program using intra-block sinking of motion candidate instructions, as shown in FIGS. 15A-15C.

Please amend paragraph [0064] of the specification at page 16 as follows:

[0064] In one embodiment, process block 588 describes intra-block hoisting. In such an embodiment, motion candidates, excluding both AWAIT operations and ADVANCE operations, are hoisted in the basic blocks, which contain AWAIT operations as high as possible without violating the dependence graph. In one embodiment, an instruction ~~is~~ that is hoisted outside of an outmost critical section is no longer regarded as a motion candidate. For example, as illustrated with reference to FIGS. 15A-15C, instructions (667/668), which are hoisted or sunk outside an outmost ADVANCE operation 664 or outmost AWAIT operation 662, are no longer considered as either sink candidates or hoist candidates. Once code motion is performed on the CFG loop, a modified CFG loop is formed, which may be used to form program-threads of a parallel version of the sequential application program.